Automated First-Order Theorem Proving in Software Engineering

Johann Schumann
RIACS / NASA Ames
schumann@ptolemy.arc.nasa.gov

June 2000

Introduction

- formal methods in software engineering $\surd$
- formal methods require tools
  - automatic
  - powerful
  - trustworthy
  - usable

Application Areas for formal methods [tools]

- throughout the entire SW life-cycle
- where?
  - Verification
  - Synthesis of
    - code
    - designs
  - software reuse
  - debugging/testing
  - . . .

Inference Systems

- inference system as a "kernel" of formal methods tools
- model checkers (SMV, SPIN)
  - successful for hardware
  - software ?
  - can you trust a MC? (no proof)
- interactive theorem provers (HOL, PVS, Isabelle, . . .)
  - too interactive
  - require specialists

Inference Systems II

- no prover
  - probably the best if you *know* what to do
- symbolic algebra systems (Mathematica,. . .) and similar systems
  - good in math, bad in reasoning
  - correct? ($(x * y)/x \Rightarrow y$)
- automated theorem provers for first order logic
  - currently restricted “more by general usability than by raw deductive power” [Kaufmann,98]
  - can they be used? / what has to be done? (this tutorial!)

Overview

1. Introduction
2. **Logic Foundations**
3. Proof Tasks and their Characteristics
4. Case Studies
   (a) logic-based component retrieval
   (b) synthesis of scientific software
   (c) verification of cryptographic protocols
5. Requirements and Techniques
6. Conclusions

Logical Foundations

- Predicate Logic
- Model Theory
- Formal Systems
- Theorem Proving
  - resolution-style provers
  - tableau-style provers
- strengths and weaknesses of ATPs

First Order Predicate Logic

- defined over alphabet of:
  - variables, constants: $X, Y, a, 999, []$
  - syntactic *function symbols*: $f(t_1, \ldots, t_m)$, $\texttt{cons}(t_1, t_2)$
  - *predicate symbols*: $p(t_1, \ldots, p_n)$, "$=$"
  - connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - *quantifiers*: $\exists, \forall$
- Example: $\forall L \cdot (L = [\,] \vee (\exists H, T \cdot L = \texttt{cons}(H, T))$
- syntax only

Model Theory

- *interpretation* of formula over *domain of discourse* $\mathcal{D}$
- valuation function: assign values to terms, $\textsc{True}/\textsc{False}$ to predicates
- Example: $\forall L \cdot (L = [\,] \vee (\exists H, T \cdot L = \texttt{cons}(H, T))$

I_1 $\mathcal{D}$ = lists	$[\,]$ is empty list	$\texttt{cons}(H, T)$ is list constructor
I_2 $\mathcal{D}$ = traces	$[\,]$ is empty trace	$\texttt{cons}(H, T)$ is prepend element to trace
I_3 $\mathcal{D} = \mathbb{N}$	$[\,] = 0$	$\texttt{cons}(H, T)$ is add number to sum $H + \Sigma T$

- $\mathcal{F}$ is *satisfiable* if there is at least one valuation v: $v(\mathcal{F}) = \textsc{True}$
  $X + 3 > 5$ is satisfiable, but not valid; $X = 0 \wedge X = 1$ is unsatisfiable
- $\mathcal{F}$ is *valid* if $v(\mathcal{F}) = \textsc{True}$ for all v and all interpretations ($\models \mathcal{F}$)
  $\forall A, B \cdot A \times B = B \times A$ not valid (matrices!)

Formal Systems

- formal system = formal language + axioms + inference rules
- purely *syntactic*
- inference rule: e.g. modus ponens $\dfrac{\mathcal{A} \qquad \mathcal{A} \rightarrow \mathcal{B}}{\mathcal{B}}$
- $\mathcal{F}$ is a theorem ($\vdash \mathcal{F}$) if obtained from axioms by using inference rules
- important: formal system S is *sound* if $\Gamma \models \mathcal{F}$ whenever $\Gamma \vdash \mathcal{F}$
- only then (syntactic) theorem proving makes sense

Theorem Proving

- purely syntactic operations
- compared to model checking: assignment of values
- often *refutation*: show $\neg\mathcal{F}$ is unsatisfiable (i.e., $\neg\mathcal{F} \vdash \textsc{False}$)
- FOL is *semi-decidable*, i.e.,
  - there is no algorithm which says $\textsc{True}/\textsc{False}$ in all cases
  - there are algorithms which eventually say $\textsc{True}$ for a valid formula
  - these algorithms usually do not terminate on non-theorems
- *completeness*: prover eventually finds the proof
- *soundness*: prover finds no false proofs

First-order Automated Theorem Proving

- black box: $\mathcal{ATP}(\mathcal{F}, \text{parameter}) \longrightarrow \textsc{True}/\textsc{False}/\text{time-out}$
- most theorem provers: input in Clausal Normal Form (CNF)
- high complexity ($\mathcal{O}(exp)$ or worse)
- two worlds (at least) of ATP:
  - *synthetic* calculi: generate new formulas from given ones
    - resolution-based theorem provers
  - *analytic* calculi: operate on given formulas, break them down
    - tableaux-based theorem provers

Clausal Normal Form (CNF)

- specific normal form for logic formulas: contains only $\wedge, \vee, \neg$
- CNF formula is a set of $\bigwedge$'d *clauses*
- a clause is a set if $\bigvee$'d *literals* (atom or $\neg$atom)
- existential quantifiers removed by *Skolemization*: e.g.,
  $\forall X \exists Y \forall Z \cdot p(X,Y,Z) \Longrightarrow p(X, f_Y(X), Z)$
- Example:

$\mathcal{F}$	$\neg \forall X \forall Y \cdot p(X,Y) \vee p(Y,X) \rightarrow \forall V \exists Z \cdot p(V,Z) \wedge p(Z,V)$
CNF	$p(X,Y) \quad \vee \quad p(Y,X) \quad \wedge$
	$\neg p(a,Z) \quad \vee \quad \neg p(Z,a)$

CNF II

- conversion algorithm pretty standard [Loveland78,ClocksinMellish84]
- many optimizations possible
  - optimization of Skolemization: shorter Skolem functions
  - nested $\leftrightarrow$'s cause exponential size of CNF formula
  - "definitional normal form" [Eder85,Nonnengart98] avoids this
  - optimizations have *significant* influence on proof times
- "back"-transformation is possible with definitional normal form; never implemented

The Resolution Rule

- [Robinson,1965], 1978 already 25 variants
- inference rule: take two clauses and generate a new one out of them

$$\frac{L, K_1, \ldots, K_l \qquad\qquad \neg L', M_1, \ldots, M_n}{\sigma K_1, \ldots, \sigma K_n, \sigma M_1, \ldots, \sigma M_n} \text{ where } \sigma L = \sigma L'.$$

- use *unification* to obtain σ
- perform resolution step, until the empty clause [] has been obtained.
- then $\neg\mathcal{F}$ is unsatisfiable ($\mathcal{F}$ is valid)

Example

$$\begin{array}{lcll} (1) & p(X,Y) & \vee & p(Y,X) \\ (2) & \neg p(a,Z) & & \end{array}$$

Proof:

$$\frac{\dfrac{p(X,Y) \vee p(Y,X) \qquad\qquad \neg p(a,Z)}{p(Z,a)} \; \sigma = [X \backslash a, Y \backslash Z] \qquad \qquad}{} $$

$$\frac{\vdots \qquad\qquad \neg p(a,Z)}{[]} \quad \sigma = [Z \backslash a]$$

Search for the Proof

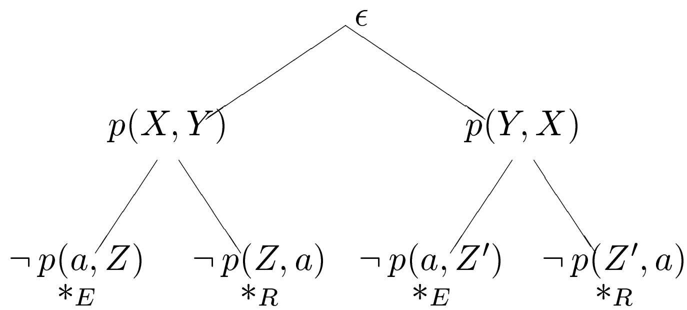
- potential for search:
  - which clauses participate in resolution
  - which literals are selected there
  - in which order to select clauses (agenda ordering)
  - which resolution rules to take
- breadth-first search
- backward and forward subsumption to reduce number of newly generated clauses

OTTER: a resolution-type ATP

- *the* classical resolution-style prover
- developed at Argonne Natl. Labs (Bill McCune)
- implemented in C
- many inference rules and parameters with "auto-mode"
- reasonably good CNF transformation
- applications mainly in mathematics

Tableau-based ATP: Model Elimination

- ME [Loveland78]
- start rule
- extension rule
- reduction rule

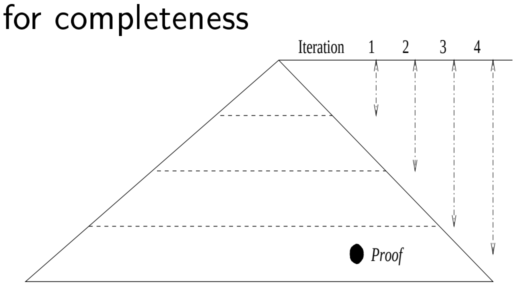


- Example: $\begin{array}{ccc} p(X,Y) & \vee & p(Y,X) & \wedge \\ \neg p(a,Z) & \vee & \neg p(Z,a) & \end{array}$

- Substitutions: $X\backslash a, Y\backslash a, Z\backslash a, Z'\backslash a$

Search for the Proof

- potential for search:
  - literal selection: which literal to take in the current clause
  - clause selection: extension into which clause
- PROLOG-style depth-first, left-to-right search
- iterative deepening for completeness



SETHEO: SEquential THEOrem prover

- developed at the Automated Reasoning Group in Munich, Germany
- implemented in C (UNIX, Linux) and PROLOG (preprocessing)
- many extensions for pruning the search space
- iterative deepening over various metrics
- parallel systems: PARTHEO, RCTHEO, SPTHEO, SiCoTHEO, p-SETHEO
- winner on CADE prover competitions (CASC)
- `http://wwwjessen.in.tum.de/~setheo`

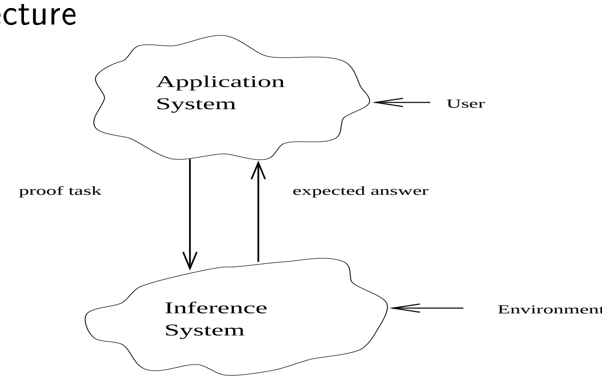
Strengths and Weaknesses of ATPs

- ATPs are not flexible with respect to logic: “FOL/CNF only”
- ATPs are fully automatic: “interactive mode is a nightmare”
- ATPs are very weak in detecting non-theorems
- ATPs are highly efficient search algorithms with many knobs to turn
- ATPs find proofs fast (or never)
- ATPs produce proofs
- ATPs: many out there (Conf: CADE (CASC), Tableaux, FOL,. . . Journal AR, Automated Deduction-A Basis for Applications (3 Vols))

Overview

Proof Tasks in Applications

Principle Architecture



- from the outside
- logic-related characteristics
- system related characteristics
- classification scheme

From the Outside I

- number of proof tasks per “session”
  - 1–10 for verification
  - 100’s to 10,000’s for component retrieval (search in a library)
- frequency
  - $\approx 50 - 100/min$ for automated online verification
    (e.g., verification of down-loadable code,proof-carrying code)
  - $\approx 1/min$ for interactive systems
  - $0.1/min - 0.01/min$ for non-interactive, batch-like verification
- “results-while-u-wait” ?

From the Outside II: size and syntactic richness

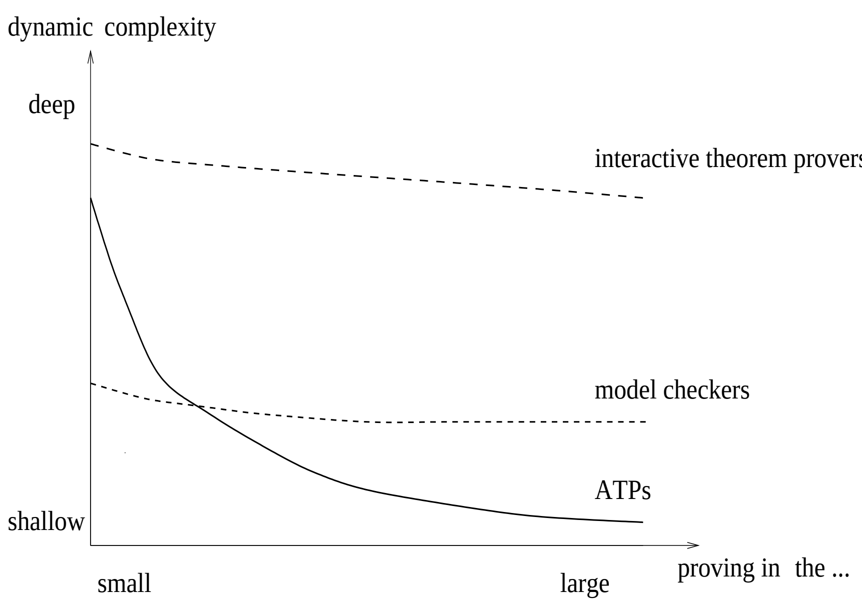
- size of the formulas
  - even small formulas can be very hard to prove
  - large formulas might contain redundancies and unused parts ($\rightarrow$ simplification)
- complexity of terms and syntactic richness
  - no function symbols (data logic): problem is decidable
  - finite domains: problem is decidable
  - rich formulas can have internal structure useful to guide the ATP
  - function symbols with large arity often produce hard-to-find proof

"Complexity"

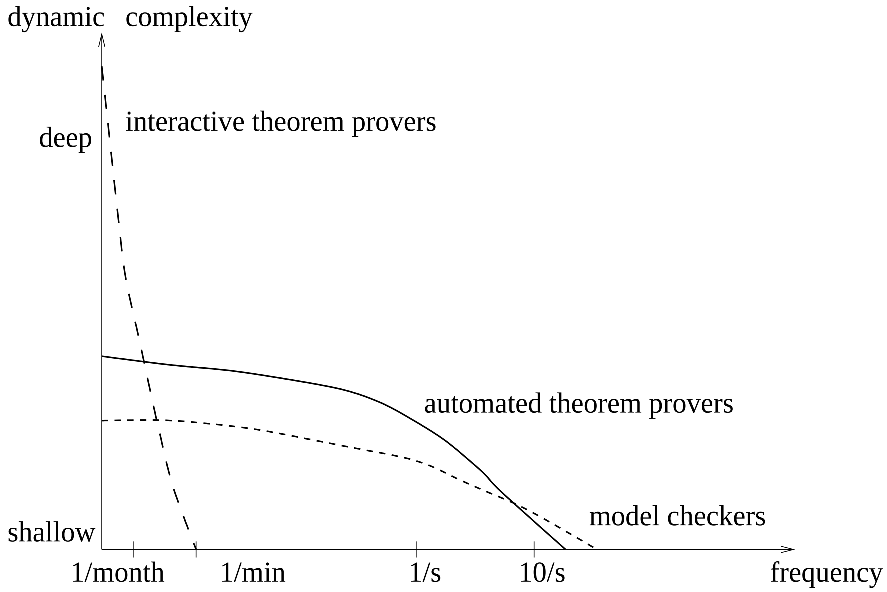
"How difficult it is to find a proof?"

- *shallow*: proof is easy to find (simple structure), although it might be buried under tons of useless information
- *deep*: complex proof structure, hard to find

"Complexity" vs. Size



"Complexity" vs. Frequency



Logic-related characteristics

- which logic?
- ratio of theorems vs. non-theorems
  "ATPs usually only can detect theorems"
- semantic information?
- expected answer
  - TRUE/FALSE?
  - answer substitution, e.g. query $\exists X \cdot p(X)$
    could returns $X = a \vee b$
  - which axioms and hypotheses have been used
  - proofs, human-readable(!) proofs

Classification Table

"start evaluating an application by filling out the classification table"

Short Table			
category	value		
deep/shallow	Shallow	Medium	Deep
number	Small	Medium	Large
size & richness	Small	Medium	Large
answer-time	Short	Medium	Long
distance	Short	Medium	Long
extensions	Y/N	which?	
validity	XX % non-theorems		
answer	TRUE/FALSE	proof	other
semantic info	Y	some	N

Overview

1. Introduction
2. Logic Foundations
3. Proof Tasks and their Characteristics
4. **Case Studies**
   (a) logic-based component retrieval
   (b) synthesis of scientific software
   (c) verification of cryptographic protocols
5. Requirements and Techniques
6. Conclusions

Case Study: Component Retrieval

- Goal: find components in a reuse library
- produce a "usable" prototype:
  - "You must find the component before you can re-use it"
  - "You must find the component faster than you can re-build it"
- deduction-based component retrieval (NORA/HAMMR)
  - attach pre- and post- conditions to the library components
  - query in form of $(\mathit{pre}_q, \mathit{post}_q)$
  - construct proof task for each retrieval operation
  - use deductive methods
- Joint work with B. Fischer [FischerSchumann98,SchumannFischer98]

Requirements

- repository = code + VDM/SL pre-post-conditions (usability)
- large repository results in *many* proof tasks (10,000's)
- "results-while-u-wait"
- ATP and logic machinery must be hidden from user

Example

- Query:
  $QUERY(x : List)\ y : List$
  $\textsc{pre}\quad x \neq []$
  $\textsc{post}\quad \exists i : Item, z : List \cdot x = [i]\hat{}z \wedge y = z\hat{}[i]$

- Candidate:
  $tail(l : List)\ m : List$
  $\textsc{pre}\quad l \neq []$
  $\textsc{post}\quad \exists i : Item \cdot l = [i]\hat{}m$

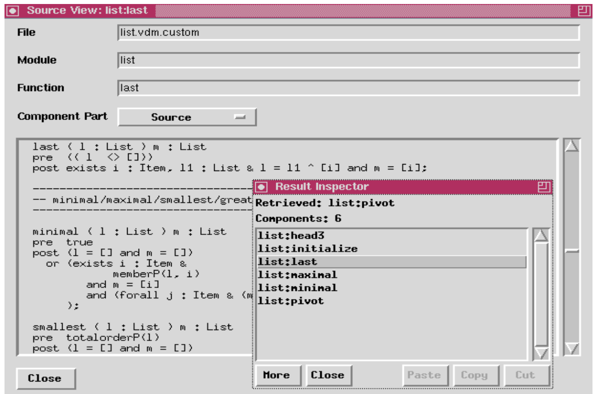
- proof task of the form: $(pre_q \Rightarrow pre_c) \wedge (pre_q \wedge post_c \Rightarrow post_q)$

- proof found $\equiv$ component can be retrieved

Proof Tasks: Characteristics

category	value		
deep/shallow	**Shallow**	**Medium**	Deep
size & richness	Small	**Medium**	Large
number	Small	Medium	**Large**
answer-time	**Short**	Medium	Long
distance	**Short**	Medium	Long
extensions	equality,sorts		
validity	10–15 % theorems		
answer	TRUE/FALSE		
semantic info	**Y**	some	N

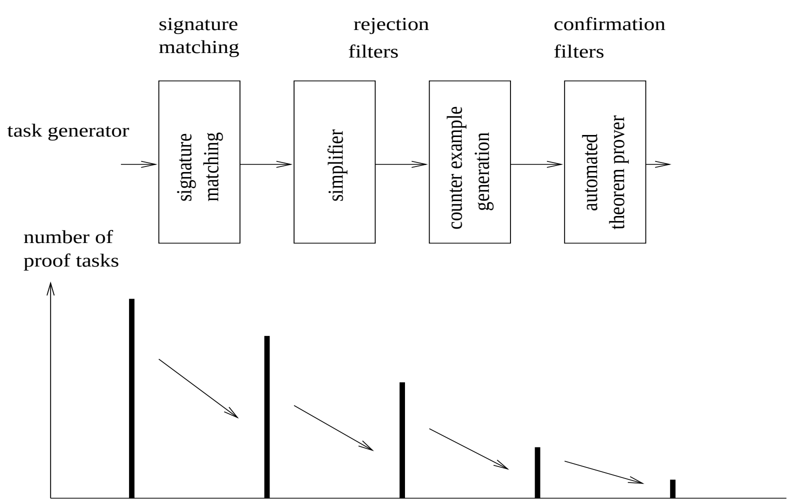
System Architecture: GUI



- Easy usability: start, stop, zoom, browser
- hiding ATP evidence
- filter pipeline

System Architecture: Filter Pipeline

Goal: drastically reduce number of proof tasks for the ATP



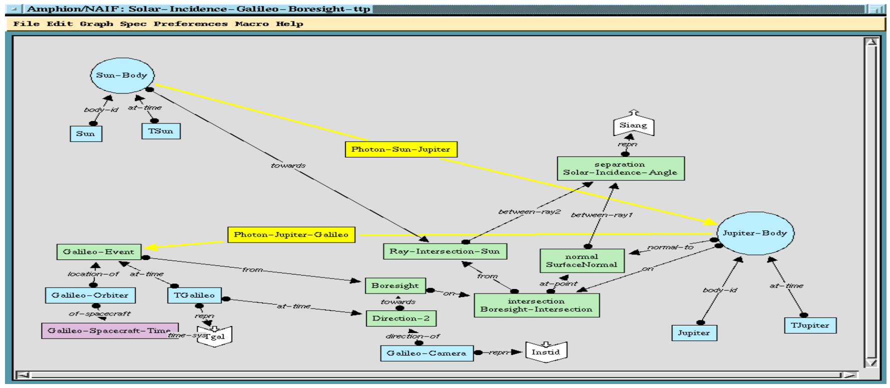
Experiments

- library of 119 specifications over lists
- full cross match for evaluation: 14161 proof tasks
- 13.1% are valid
- Results:
  - with SETHEO, we get a recall of 74.5%
  - just plug-and-play connection of ATP? NO
  - what had to be done?

Case Study: Deductive Synthesis of Astrodynamics Programs

- NAIF Fortran library of astrodynamic routines
  - well standardized
  - hard to use because of the FORTRAN names: `VXSEC(...)`
  - problem solutions can be assembled from library calls
- Goal: Given a graphical specification, synthesize the corresponding FORTRAN program
- The system: AMPHION [Lowry etal]
  - fully deductive
  - based on the SNARK FOL theorem prover (resolution-style)

Example: Specification



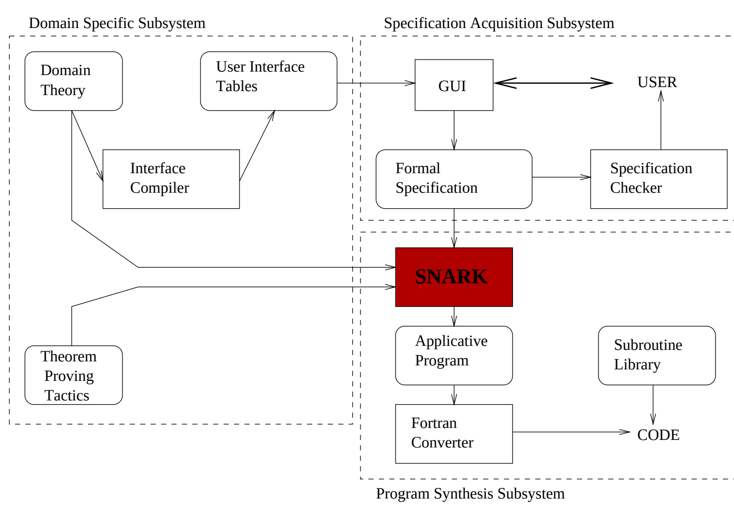
Example: Produced Code

```
  SUBROUTINE SOLAR0 (TGAL,INSTID,SIANG)
C ...
C     Input variables
  CHARACTER*(*) TGAL
  INTEGER INSTID
C     Output variables
  DOUBLE PRECISION SIANG
C ...
  CALL SCS2E ( GALILE, TGAL, ETGALI )
  CALL BODVAR ( JUPITE, 'RADII', DMY1, RADJUP )
  CALL SPKSSB ( GALILE, ETGALI, 'J2000', PVGALI )
  CALL SCE2T ( INSTID, ETGALI, TKINST )
  TJUPIT = SENT ( JUPITE, GALILE, ETGALI )
  CALL BODMAT ( JUPITE, TJUPIT, MJUPIT )
  CALL ST2POS ( PVGALI, PPVGAL )
  CALL SPKSSB ( JUPITE, TJUPIT, 'J2000', PVJUPI )
C ...
  CALL SURFNM ( RADJUP(1), RADJUP(2), RADJUP(3), P, PP)
  CALL MTXV ( MJUPIT, P, XP )
  CALL MTXV ( MJUPIT, PP, XPP )
  CALL VADD ( PPVJUP, XP, V0 )
  CALL VSUB ( PPVSUN, V0, DV0PPV )
  SIANG = VSEP ( XPP, DV0PPV )
  RETURN
  END
```

Characteristics

category	value		
deep/shallow	Shallow	**Medium**	Deep
size & richness	Small	**Medium**	**Large**
answer-time	Short	**Medium**	Long
distance	**Short**	Medium	Long
extensions	equations,λ-terms		
validity	100 % theorems		
answer	variable substitutions explanations		
semantic info	**Y**	some	N

Amphion: System Architecture



Amphion: Why does it work?

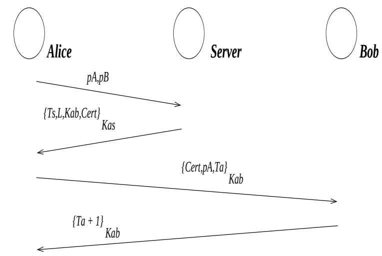
- short distance between input specification and synthesized code
- linear code (i.e., no loops or recursion)
- rewriting and simplification
- decision procedures for specific operations
- answer substitution = functional program
- additional information used to generate explanations
- adapted to other domains: fluid dynamics, navigational software

Case Study: Verification of Authentication Protocols

- Authentication protocols and cryptographic protocols widely in use
  - WWW
  - e-commerce
  - cellular phones, etc
- Authentication protocol (AP): partners must be correctly identified
- high vulnerability
  - bugs in most protocols
  - weak/bad encryption, etc.
- Verification important
- many approaches; here BAN-logic [Burrows, Abadi, Needham 89]

Example: The Kerberos Protocol

- protocol = sequence of messages
- formalized in BAN logic
  (multi-sorted modal logic)
- custom logic
- $pB \mid\equiv pA \mid\sim \{T_a, pA \overset{K_{ab}}{\leftrightarrow} pB\}_{K_{ab}}$
- defined by ca. 10 inference rules
- typical proof task: $pA \mid\equiv pA \overset{K}{\leftrightarrow} pB$



Requirements

- automatic operation
- input and proofs in BAN-logic

category	value		
deep/shallow	Shallow	**Medium**	Deep
size & richness	Small	**Medium**	Large
answer-time	Short	**Medium**	Long
distance	Short	**Medium**	Long
extensions	finite messages		
validity	80%- 90 % theorems		
answer	**readable proof in BAN-logic**		
semantic info	Y	some	**N**

Example: manual Proof

As an example for a proof in the BAN logic, let us again consider the Kerberos protocol. We want to show that

$$pB \models pA \models pA \overset{K_{ab}}{\leftrightarrow} pB \tag{1}$$

holds, after messages 1–3 have arrived. Before message 3 has arrived, we already know from a previous proof task that

$$pB \models pA \overset{K_{ab}}{\leftrightarrow} pB. \tag{2}$$

By the inference rule "message-meaning" and with idealized message 3 (second part) of the protocol, we obtain

$$pB \models pA \mid\!\sim \{T_a, pA \overset{K_{ab}}{\leftrightarrow} pB\}_{K_{ab}}. \tag{3}$$

Since, by assumption $pB \models \#T_a$, we have (if a part of a message is believed to be fresh, then the entire message is)

$$pB \models \#(\{T_a, pA \overset{K_{ab}}{\leftrightarrow} pB\}_{K_{ab}}). \tag{4}$$

Finally, by (3) and (4) and "nonce-verification", we can prove our theorem (1). q.e.d.

Example: Output of PIL/SETHEO

Theorem 1. $query \qquad (\vdash\ pB \models\ pA \models\ pA \overset{K_{A,B}}{\leftrightarrow} pB).$

Proof (by SETHEO). We show directly that

$$query. \tag{5}$$

Because of $message\_3$

$$\vdash\ pB \triangleleft\ (\{\{\{T_S, pA \overset{K_{A,B}}{\leftrightarrow} pB\}\}_{K_{B,S}}, \{\{T_A, pA \overset{K_{A,B}}{\leftrightarrow} pB\}\}_{K_{A,B}}\}). \tag{6}$$

Because of $query\_3$

$$query \leftarrow \vdash\ pB \models\ pA \models\ pA \overset{K_{A,B}}{\leftrightarrow} pB. \tag{7}$$

Because of $break\_up\_message$

$$\vdash\ P \models\ Q \models\ X \leftarrow X \sqsubseteq Y \wedge \vdash\ P \models\ Q \models\ Y. \tag{8}$$

Because of $nonce\_verification$

$$\vdash\ P \models\ Q \models\ X \leftarrow \vdash\ P \models\ Q \mid\!\sim\ X \wedge \vdash\ P \models\ \#X. \tag{9}$$

Because of $freshness \vdash P |\equiv \#M_i \leftarrow \vdash P |\equiv \#\{M1, \ldots, M_n\}$.
Because of $assumption\_11 \vdash pB |\equiv \#T_A$. Therefore

$$\vdash pB |\equiv \#(\{T_A, pA \overset{K_{A,B}}{\leftrightarrow} pB\}). \tag{10}$$

Because of $message\_meaning$

$$\vdash P |\equiv Q |\sim X \leftarrow \vdash P \triangleleft \{X\}_K \land \vdash P |\equiv Q \overset{K}{\leftrightarrow} P. \tag{11}$$

Because of $conjecture\_2$

$$\vdash pB |\equiv pA \overset{K_{A,B}}{\leftrightarrow} pB. \tag{12}$$

Because of $sees\_components \vdash P \triangleleft M_i \leftarrow \vdash P \triangleleft \{M_1, \ldots, M_n\}$. Hence by (6)
$\vdash pB \triangleleft \{\{T_A, pA \overset{K_{A,B}}{\leftrightarrow} pB\}\}_{K_{A,B}}$. Hence by (11) and by (12)
$\vdash pB |\equiv pA |\sim (\{T_A, pA \overset{K_{A,B}}{\leftrightarrow} pB\})$. Hence by (9) and by (10)
$\vdash pB |\equiv pA |\equiv (\{T_A, pA \overset{K_{A,B}}{\leftrightarrow} pB\})$. Hence by (8) $\neg\, query$. Hence by (7) $query$. Thus we have completed the proof of (1).

q.e.d.

Overview

1. Introduction
2. Logic Foundations
3. Proof Tasks and their Characteristics
4. Case Studies
   (a) logic-based component retrieval
   (b) synthesis of scientific software
   (c) verification of cryptographic protocols
5. **Requirements and Techniques**
6. Conclusions

Requirements and Techniques

- Is FOL automated theorem proving suitable at all?
- How to connect an ATP?
- How to handle logic extensions?
- How to get results?
- How to handle non-theorems?

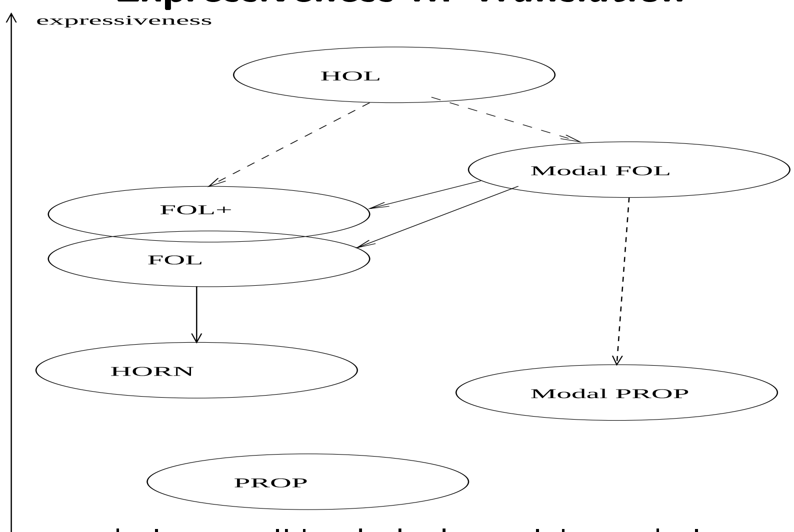
Expressiveness

Can your input logic be handled by the ATP?

Can I transform my favorite logic into FOL?

- often Higher-order logic is not that “high”:
  $\forall P \in \{send, receive\} \cdot \forall Data \cdot \textsf{correct}(Data) \rightarrow P(Data)$
- finite domains, finite state spaces make things easier
- have a close look at higher order quantifier positions
- is translation possible?

Expressiveness II: Translation



- solid line: translation possible, dashed: partial translation
- back-translation?

Hilbert-style T-Transformation

- "Meta" approach: for $\mathcal{F}$ in logic M we define first order predicate $\mathbf{T}(..)$

$$\mathbf{T}(\mathcal{F}') \equiv \textsc{True} \Leftrightarrow \mathcal{A} \vdash_M \mathcal{F}$$

- formulas become terms
- inference rules become FOL formulas:

$$\frac{\mathcal{F}_1 \quad \ldots \quad \mathcal{F}_n}{\mathcal{G}}$$

  is translated into $\mathbf{T}(\mathcal{F}_1') \wedge \ldots \wedge \mathbf{T}(\mathcal{F}_n') \rightarrow \mathbf{T}(\mathcal{G}')$.

Hilbert-style T-Transformation

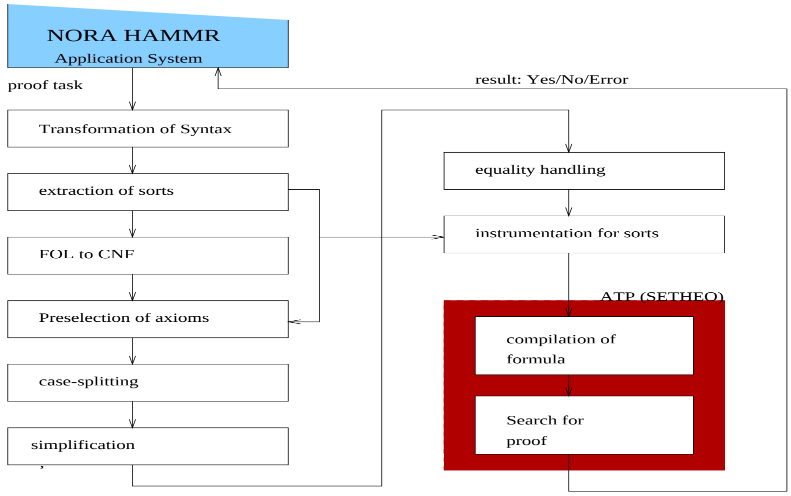
- often convenient
- can induce huge search spaces
- can often be combined with ordinary proof procedures [Ohlbach98]
- can cause problems with quantifiers

Connecting the ATP

robust and reliable system architecture for

- reading in / preparing proof task
- starting the prover(s)
- assembing/analyzing the result (SUCCESS)
- stopping the ATP
- cleaning up

System architecture



System architecture for the reuse case study

Extensions: Induction

Induction often required during program verification (recursive data structures, time-lines)

- induction is inherently higher order
  - variable(s) to perform induction ("induction over i")
  - induction scheme ("$n \to n + 1$")
  - induction hypotheses and additional lemmata
  - "base-case" and "step-case"
- can induction be performed by a first order ATP?
- many proof obligations are fairly "standard"

Ways to do Induction

- additional lemmata: e.g., $\forall l$: list $\cdot \exists l, m, r$: list $\cdot\, l = l\,^\wedge m\,^\wedge r$
- splitting up into several proof tasks:

$$\mathcal{F}([])$$
$$\forall l : \text{list} \cdot \forall i : \text{item}, l_0 : \text{list} \cdot \mathcal{F}(l_0) \wedge l = [i]\,^\wedge l_0 \rightarrow \mathcal{F}(l)$$

- "Poor Man's Induction":

$$\mathcal{F}([])$$
$$\forall l_0 : \text{list} \cdot \forall i : \text{item} \cdot \mathcal{F}([i]\,^\wedge l_0)$$

Induction: Experimental Results

SW-reuse	number of tasks				
Method	total	solved	failed	error	%-solved
axioms only	1838	1039	730	69	**56.5%**
w/lemmas	1838	1271	498	69	**69.2%**
case-splitting	1838	1235	487	114	**67.2%**
base	1838	1658	111	69	90.2%
step	1838	1235	487	114	67.2%
poor man's	1838	1302	467	69	**70.8%**
base	1838	1658	111	69	90.2%
step	1838	1302	467	69	70.8%

- Poor-man's often better, because formulas are smaller
- cannot be complete, but good results in practice

Sorts

- most proof tasks in verification are sorted: $\forall x$: nat $\cdot$ $\forall l$: list ...
- types in general are undecidable
- if sort hierarchy is upper semi-lattice, then sorted unification is unitary.
- this case is the interesting one
- mapping into ATP:
  - sorts as predicates: *huge* search space
  - sorted unification: need to modify prover
  - pre-compilation into terms

Compilation of sorts into terms

- checking of sorts done by unification
- Example (many-sorted logic): $\forall X$: nat $\cdot$ $p(X)$
  compiled into `p(X,nat)`
- extension to tree and DAG structure possible [Mellish,88]
- tools available but some intricacies

How to get results out of the prover?

- Preselection of axioms
- Simplification
- Control of the Prover

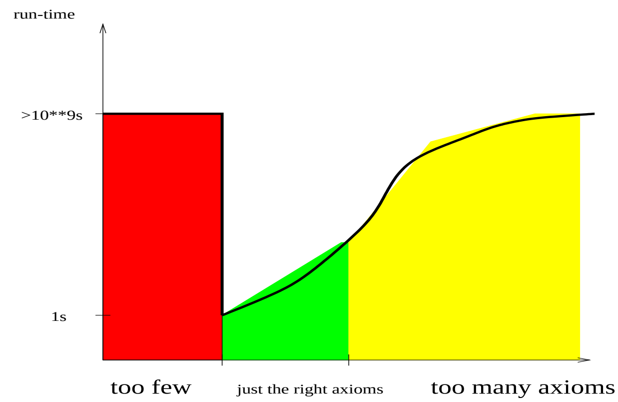
Preselection of Axioms

- Domain theory is described by axioms
- all operators/functions are defined by axioms
- Example: lists with *cons* and *app*end:

$$
\begin{array}{ll}
(1) & \forall X, Y : \text{list } \forall I, J : \text{item} : cons(I, X) = cons(J, Y) \rightarrow I = J \wedge X = Y \\
(2) & \forall X : \text{list } \exists Y : \text{list } \exists Z : \text{item} : X = cons(Z, Y) \vee X = [\,] \\
(3) & \forall L : \text{list } \forall X : \text{item} : cons(X, L) \neq [\,] \\
(4) & \forall X : \text{list } \forall Y : \text{list } \forall I : \text{item} : app(cons(I, L), X) = cons(I, app(L, X)) \\
(5) & \forall L : \text{list} : app([\,], L) = L \\
(6) & \forall X : \text{list } \forall Y : \text{list } \forall Z : \text{list} : app(app(X, Y), Z) = app(X, app(Y, Z)) \\
(7) & \forall X : \text{list } \forall Y : \text{list} : app(X, Y) = [\,] \leftrightarrow X = [\,] \wedge Y = [\,] \\
(8) & \forall L : \text{list} : app(L, [\,]) = L
\end{array}
$$

Axioms and ATP

- axioms can span a *considerable* search space.
- especially transitivity is harmful: $\forall X, Y, Z \cdot p(X,Y) \wedge p(Y,Z) \rightarrow p(X,Z)$
- too few axioms $\rightarrow$ no proof
- too many axioms $\rightarrow$ no proof
- needed:
  what are the right axioms?



Preselection of Axioms

- in general: undecidable
- good approximation [Dahn,Schellhorn/Reif,Fischer]:
  - use hierarchical theories (e.g., one for each group of operators)
  - hierarchy forms DAG or tree
  - select only those sub-theories which
    - are used in the conjecture
    - are dependent from already selected theories
- reuse case study (% solved problems):

no axioms	46.3%
all axioms	55.9%
preselection	69.2%

Simplification

- most *generated* proof tasks contain redundant parts
- symbolic algebra systems and interactive TPs:
  many person-years spent on *simplifiers*
- ATP: usually *no* built-in simplifiers
- Reason: benchmarks (TPTP) library contains no redundancies

Simplification II

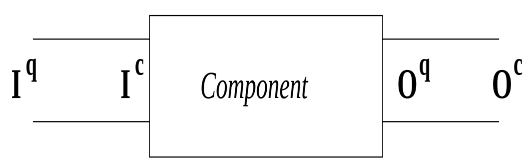
- case studies show: simplification
  - is extremely important
  - can solve simple problems (26% in reuse case study)
  - can detect many non-theorems (later)
  - reduces size of formula
  - reduces processing time (compiling, loading,. . .)
  - increases number of solved tasks considerably
- here: some powerful, yet easy to perform simplifications (preprocessing)

Syntactic Simplification

- logic simplification (of course): $\mathcal{A} \wedge \textsc{True} \Rightarrow \mathcal{A}$
  - important when considering specific cases (e.g., induction)
  - $X = [] \wedge (X \neq [] \wedge \mathcal{F}) \ldots$
- removal of definitions $neq(X, Y) \leftrightarrow \neg equal(X, Y)$
  - usually shortens proofs: no intermediate steps to expand/contract definitions
  - can have dramatic effects (both ways)
  - in practice: only expand 1:1 definitions

Syntactic Simplification II

- removal of simple equations of the form $X = t$
  - example (SW reuse):



$$\forall I_1^q, \ldots, I_n^q, O_1^q, \ldots, O_m^q \cdot \forall I_1^c, \ldots, I_n^c, O_1^c, \ldots, O_m^c \cdot$$
$$(I_1^q = I_1^c \wedge \ldots \wedge I_n^q = I_n^c \wedge O_1^q = O_1^c \wedge \ldots \wedge O_m^q = O_m^c$$
$$\rightarrow \mathcal{F})$$

    reduces to $\mathcal{F}$ with variable renamings
  - for many proof tasks: run-time reduction by factor of 10

Semantic Simplification

- using a set of rewriting rules extracted from domain theory
- not necessarily confluent
- examples:
  $\forall H, T \cdot \mathtt{cons}(H, T) \neq []$
  $\forall H, T \cdot \mathtt{hd}(\mathtt{cons}(H, T)) = H$
- powerful: application of induction schemas plus simplification
- powerful: unrolling of recursive definitions plus simplification
- result (SW reuse): more than 40% of non-valid proof tasks eliminated

Control of the Prover

Requirements:

- *usability*: control hidden from the user
- *smoothness*: similar behavior on similar proof tasks
- *speed*: short answer times
- *practical completeness*: "we are slow, but we get more tasks solved"

Reality: ATPs have 100's of user-selectable parameters,
some of them known only to the developers of the system
Reality: . . . and even forgotten by the developers

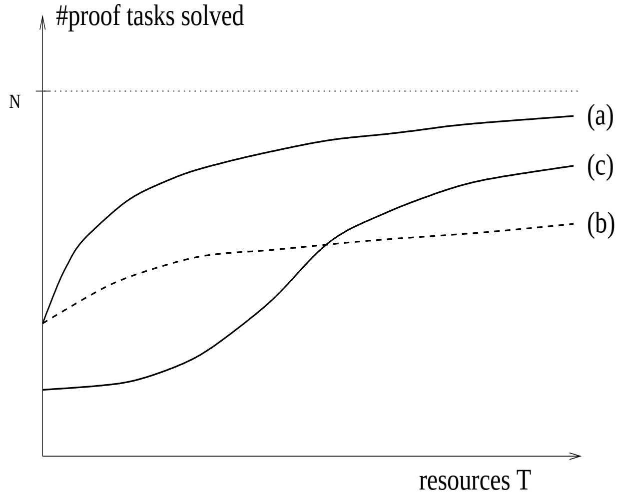
Speed vs. Practical Completeness

number of tasks solved with $t_p < t$

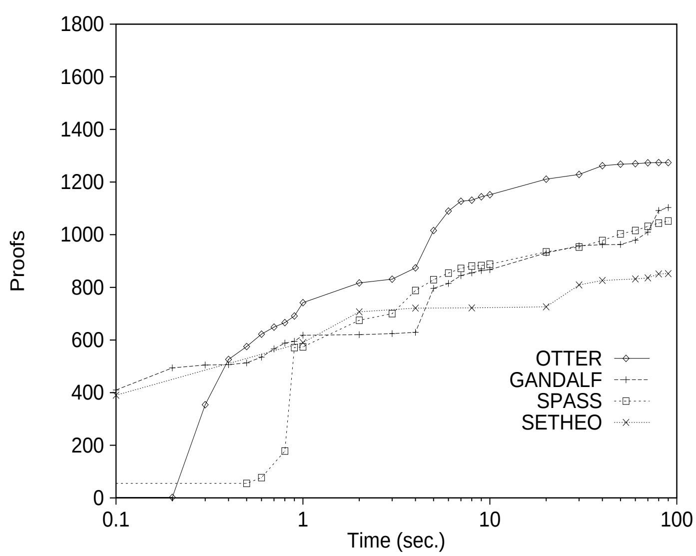
(a) the ideal case

(b) aim at short answer times

(c) aim at solving as many tasks as possible, but can have larger run-times



Runtime Behavior of well-known provers



reuse case study (from [Fischer,2000])

Smoothness

- similar proof tasks should result in similar run-times
  - unfortunately not
  - Note:
    "changing one $\neg$ can change validity"
- similar parameter settings should result in similar behavior
  - unfortunately not
- can often help: try out different settings in parallel
  - competitive parallelism
  - network of workstations or schedule
  - usually good results

Handling of Non-theorems

- FOL is undecidable
- $\vdash \mathcal{F}$ prover eventually stops with "SUCCESS"
- $\not\vdash \mathcal{F}$ prover almost never stops
- Try $\neg\mathcal{F}$?
  - usually $\neg\mathcal{F}$ doesn't do the job either
  - we have: valid, satisfiable, unsatisfiable
- many applications produce large numbers of non-theorems.
  E.g., SW reuse: only 13.1% of proof tasks (1838 of 14161) are valid

Non-theorem Detection by Simplification

- use simplification on formula
- try to reduce to $\textsc{True}$ or $\textsc{False}$
- combine with induction/definition unrolling: e.g.,
  $\mathcal{F}[X\backslash[]] \ \wedge \ \forall H$: item $\forall T$: list $\cdot \mathcal{F}[X\backslash\mathtt{cons}(H,T)]$
- SW reuse: 49.5% of non-theorems detected in $< 2s$ sun ultra-sparc

Generation of Counterexamples

- only possible for finite domains
- systems: Finder [Slaney], ANLDP [McCune], ...
- problem (big): how to make the domains finite:
  - Abstraction
  - Approximation
- *very* difficult in practice

Conclusions

- ATPs can be successfully applied
- it is no plug-and-play
- ATP developers start to work on applications
- applications needed to drive ATP applications and applicability
- that is You!

Bibliography/WWW-pages

- J. Schumann, *Automated Theorem Proving in Software Engineering*. Habilitation Thesis, Technische Universität München, 2000. [ask me is you want a copy]
- W. Bibel and P. Schmitt (eds.) *Automated Deduction: A Basis for Applications*. Kluwer, 1999. (3 volumes)
- S. Hölldobler (ed.) *Intellectica and Computational Logic: Papers in Honor of W. Bibel*. Kluwer, 2000.
- `http://www-formal.stanford.edu/clt/ARS/systems.html`
- `http://www.comlab.ox.ac.uk/archive/formal-methods.html`
- `http://www.ase.arc.nasa.gov`
- and many more